BLOCKMAKER LIVE

Guía de ejercicios

Módulo 7

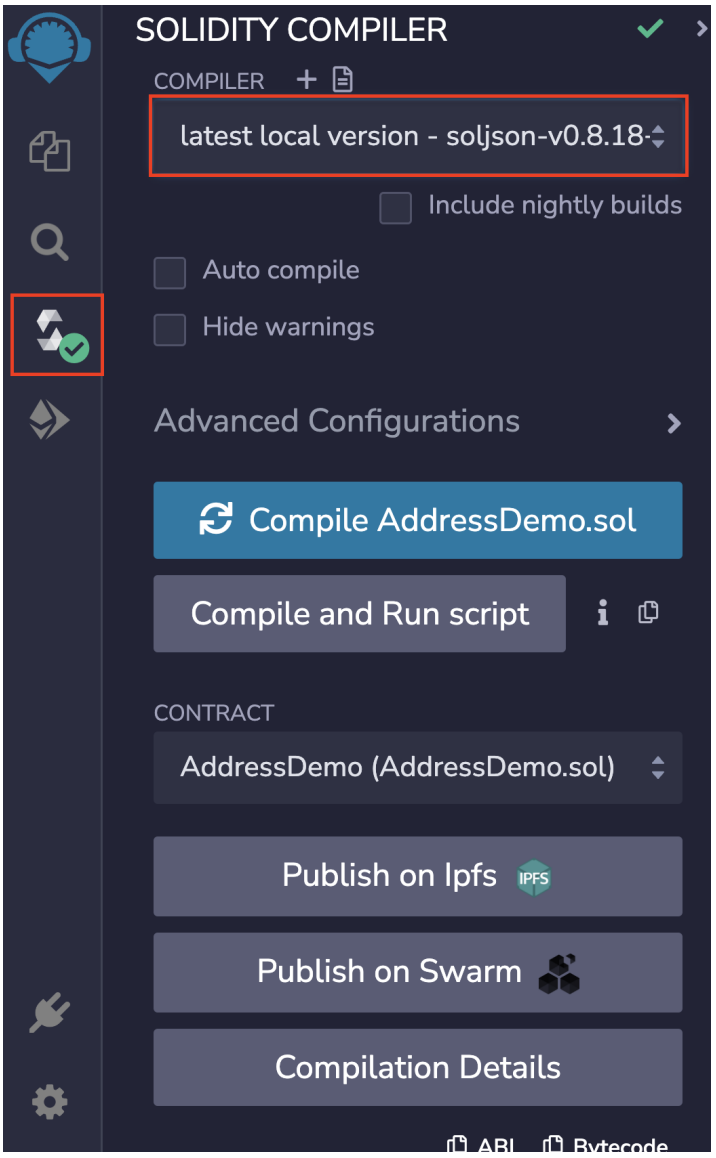
Recordatorios

1.A continuación, os dejamos un enlace a modo de bibliografía y de utilidad para consultar cualquier duda en materia de Solidity (si lo necesitáis, usar la función de traducir página al español):

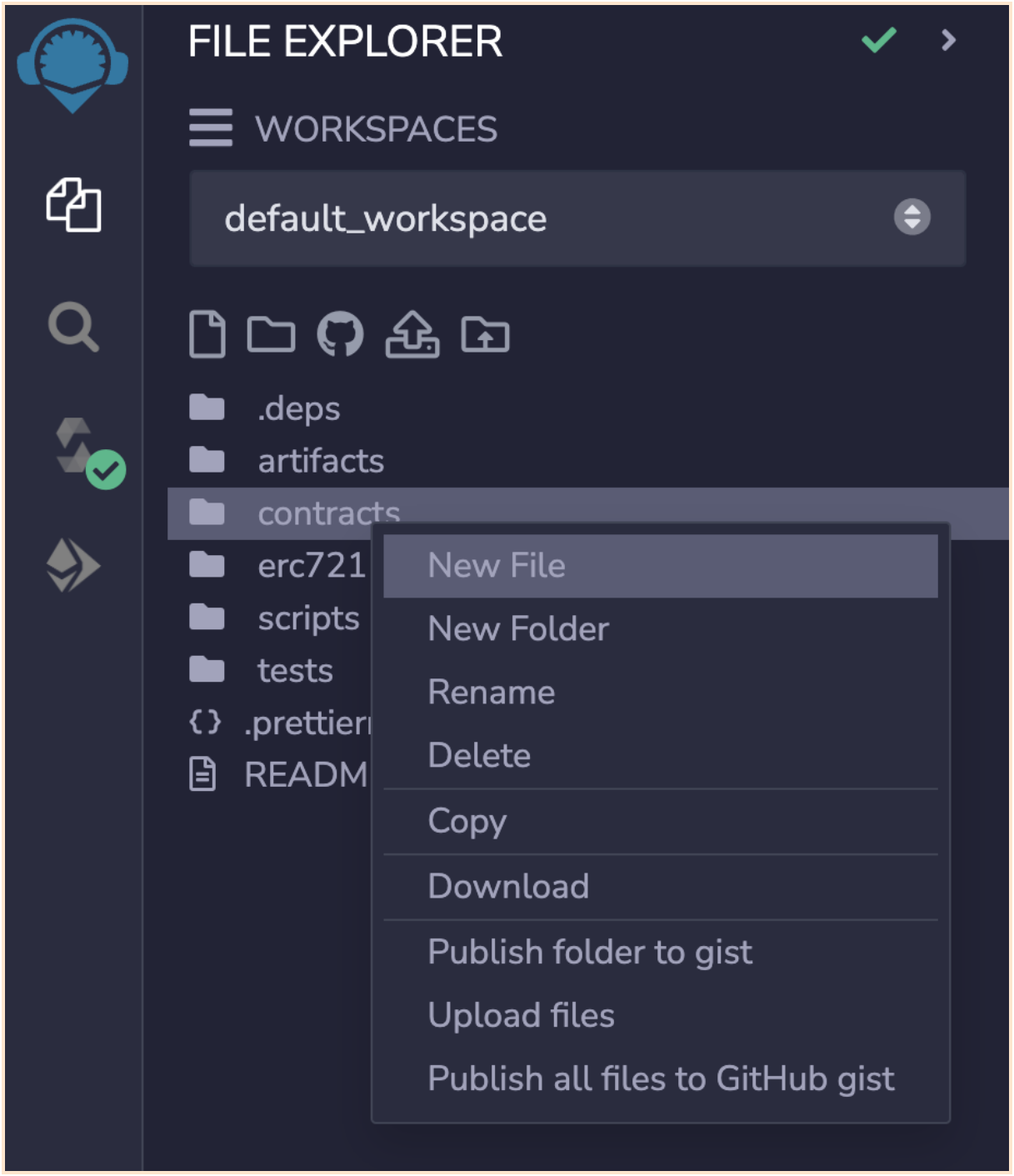
- https://docs.soliditylang.org/en/v0.8.21/grammar.html

2.Para realizar estos ejercicios vamos a utilizar el compilador online de Remix http://remix.ethereum.org/

RECUERDA 1: Desde la página http://remix.ethereum.org/ vamos a irnos a la tercera sección de la izquierda y vamos a cambiar la versión del compilador. Para ello, seleccionaremos cualquiera superior a la 0.8.0:



RECUERDA 2: Para crear un nuevo contrato, nos iremos a la primera sección de la izquierda y colocaremos en la carpeta “contracts” -> “New File” y le pondremos una nombre con la extensión “.sol.



Paso 0. Crear el contrato

Crea un nuevo contrato Solidity con el nombre "**BancoDescentralizado**."

Paso 1. Introducir cabecera contracto

Copiar al inicio del contrato inteligente.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;
```

Paso 2. Declarar “contract” BancoDescentralizado

Creamos el **contract BancoDescentralizado** (recuerda que se declara como si fuera una clase y se le añaden las llaves para introducir dentro todo el contenido).

Paso 3. Crear variable propietario

Define una **variable de estado llamada "propietario"** de tipo **address** y haz que sea **privada (private)**. Esta variable almacenará la dirección del propietario del contrato.

Paso 4. Crear constructor

Crea una **función constructora (constructor) que establezca el valor de "propietario" como la dirección que despliega el contrato**.

Paso 5. Crear mapping

Crea un mapeo llamado "**saldos**" que mapea **la dirección de Ethereum del usuario** (de tipo **address**) con su **saldo** (de tipo **entero**). Este mapeo debe ser **privado (private)**.

Paso 6. Implementar función depositar

Implementar una función **pública**, de tipo **payable** (vamos a enviar ether al contrato inteligente) llamada "**depositar" sin parámetros de entrada.**

***Descripción**:*

- Esta función debe **permitir a cualquier usuario agregar saldo a su cuenta.**
- Asegúrate**,** de que el **valor del mensaje / petición (msg.value)** sea **>0.**
- Finalmente, agrega la cantidad que ha enviado al mapping **"saldos"** del emisor del mensaje (msg.sender) con el valor en ether que ha enviado el propio usuario (msg.value).

Paso 7. Implementar función retirar

Implementar una función **pública** llamada "**retirar" (piensa en el tipo de mutabilidad).**

***Parámetros de entrada**:*

- Variable **cantidad** de tipo entero.

***Descripción**:*

- Esta función **NO debe permitir retirar fondos al propietario del contrato inteligente** (propietario del banco).
- Esta función debe **comprobar que el saldo del emisor del mensaje (msg.sender) es superior o igual (>=) a la cantidad que desea retirar.**
- El contrato debe **actualizar el saldo del usuario restándole la cantidad que desea retirar.**
- **(EXTRA 1)** Finalmente, usando la **función "transfer()", vamos a retirar saldo del contrato inteligente (banco) y se lo vamos a devolver al usuario** (esta parte la veremos en la clase de resolución de ejercicios, ya que vamos a utilizar funciones para retirar ether). De momento, copiar esto al final de la función.

```
payable(msg.sender).transfer(_cantidad);
```

Paso 8. Implementar función obtenerSaldo

Implementar una función **pública** llamada **obtenerSaldo (piensa en el tipo de mutabilidad). No precisa de parámetros de entrada.**

***Descripción**:*

- Esta función debe **devolver el saldo de la persona emisora de la petición / mensaje (msg.sender)**

(EXTRA 2). Probar el contrato inteligente en remix

Si te sientes capacitado, prueba a desplegar el contrato inteligente en remix y comprueba si puedes llamar a la función “depositar”.

NOTA: En la clase explicaremos el funcionamiento de todo el contrato inteligente.